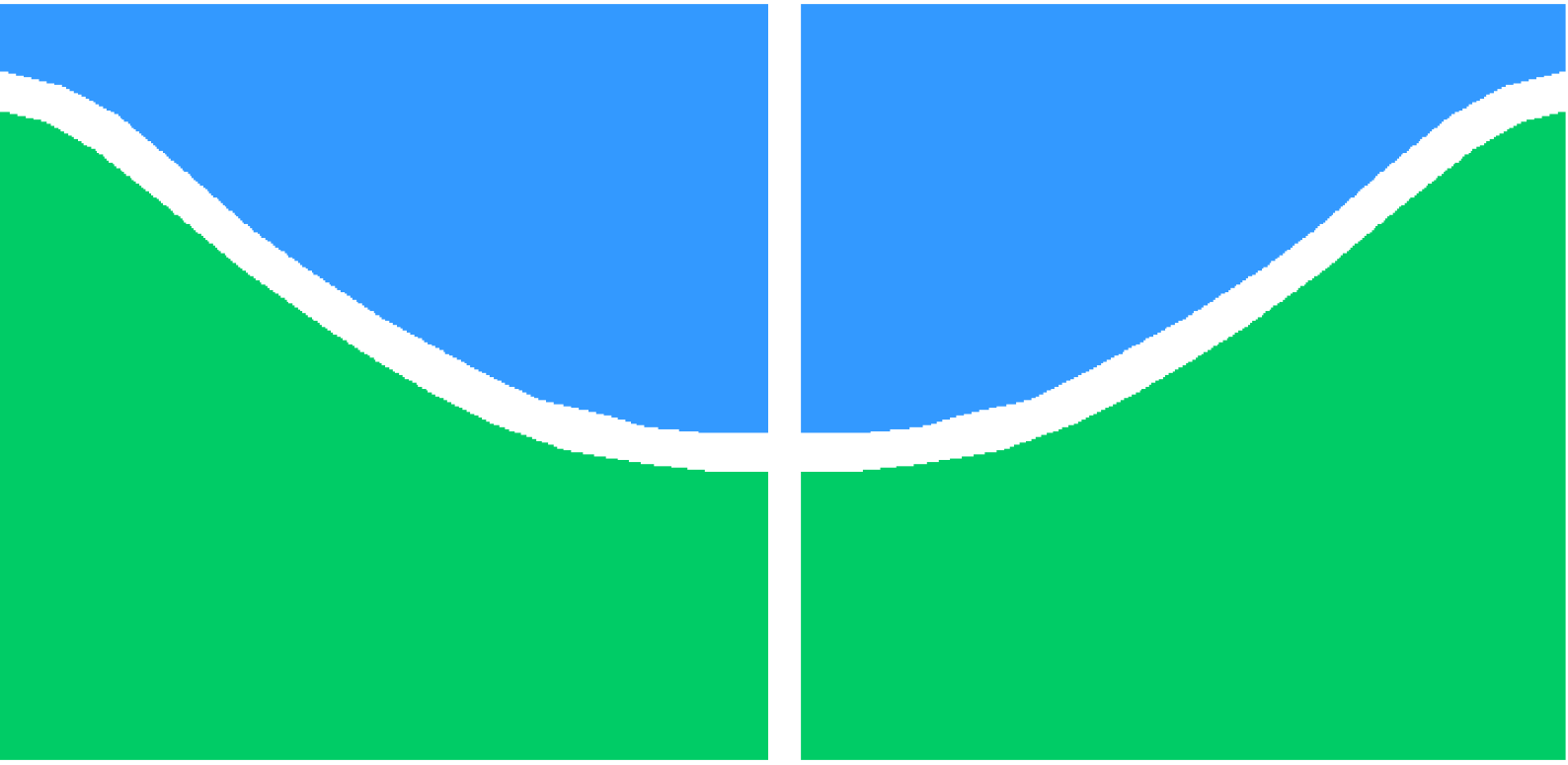

Universidade de Brasília - UnB
Faculdade UnB Gama - FGA
Engenharia de Software

Aprendizado por reforço aplicado ao ambiente *Toy Text* da plataforma *Gym*

Autor: Elmar Roberto Caixeta Filho
Orientador: Profº Matheus de Sousa Faria

Brasília, DF
2018

Elmar Roberto Caixeta Filho

Aprendizado por reforço aplicado ao ambiente *Toy Text* da plataforma *Gym*

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Profº Matheus de Sousa Faria

Coorientador: Profº Dr. Edson Alves da Costa Junior

Brasília, DF

2018

Elmar Roberto Caixeta Filho
Aprendizado por reforço aplicado ao ambiente *Toy Text* da plataforma *Gym*/ Elmar Roberto Caixeta Filho. – Brasília, DF, 2018-
53 p. : il. (algumas color.) ; 30 cm.

Orientador: Profº Matheus de Sousa Faria

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2018.

1. Aprendizado por Esforço. 2. Jogos. I. Profº Matheus de Sousa Faria. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Aprendizado por reforço aplicado ao ambiente *Toy Text* da plataforma *Gym*

CDU 02:141:005.6

Elmar Roberto Caixeta Filho

Aprendizado por reforço aplicado ao ambiente *Toy Text* da plataforma *Gym*

Monografia submetida ao curso de graduação em Engenharia de Softwareda Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 01 de junho de 2018:

Profº Matheus de Sousa Faria
Orientador

Prof Dr. Edson Alves da Costa Junior
Coorientador

Prof Dr. Fábio Macedo Mendes
Convidado 1

Brasília, DF
2018

Agradecimentos

Agradeço aos meus pais, que me deram apoio em todas as escolhas difíceis. Sou grato também aos meus amigos, que não me deixaram ser vencido pelo cansaço. Obrigado aos meus orientadores, pela paciência, inspiração e conhecimento compartilhado.

“Tudo o que temos de decidir é o que fazer com o tempo que nos é dado.“
(Gandalf, O cinzento)

Resumo

Será apresentado neste trabalho a aplicação de uma abordagem de aprendizagem por reforço, *Q-Learning*, no ambiente de treinamento e abstração de jogos, *Toy Text*, disponível na ferramenta *Gym*. O ambiente escolhido possui uma interface numérica em texto, sendo o *Taxi* como jogo usado para aplicação. Não houve modificações algorítmicas e todas as adições de parâmetros foram baseadas em padrões e conceito da própria abordagem que visam a resolução de alguns problemas conhecidos, como exemplo, o controle do processo markoviano. Fora implementado um agente com ações aleatórias para a comparação e análise dos resultados dos treinamentos.

Palavras-chaves: Aprendizado por reforço. Jogos. *Q-Learning*. Agentes.

Abstract

In this paper we will present the application of a reinforcement learning (RL) approach, *Q-Learning*, in the game training and abstraction environment, *Toy Games*, available in the *Gym* toolkit. The chosen environment has a text numeric interface, being *Taxi* as game used for analyze. There were no algorithmic modifications and all parameter additions were based on standards and concept of the approach itself that aims the resolution of some known problems such as control of the Markovian process. An agent with random actions was implemented to compare and analyze the results of the training.

Key-words: Reinforcement Learning. Games. Q-Learning. Agents.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

IA	Inteligência Artificial
AR	Aprendizagem por Reforço
MDP	*Markov Decision Process*

Lista de símbolos

γ Valor de Desconto

α Taxa de aprendizagem

r Recompensa/*Reward*

a Ação/*Action*

s Estado/*State*

Sumário

Introdução

Moldar uma inteligência artificial é procurar simular a inteligência nas máquinas de forma que elas possam agir e pensar como ou semelhante a seres humanos (KONAR, 2005). Para isso, o sistema de aprendizado por reforço (AR) passa por um processo de aprendizado baseado na interação do agente com um ambiente, podendo este ser, real, como o treinamento de robôs, ou artificial, como treinamento em jogos digitais.

Empresas como Deepmind e OpenAI foram criadas para desempenharem pesquisas em inteligência artificial (IA) publicando vários trabalhos, dentre eles alguns em aprendizagem por reforço ligado a jogos digitais, a exemplos, a aplicação de aprendizado por reforço em jogos do Atari (MNIH et al., 2013) e a criação de ferramentas que auxiliam no treinamento de agentes, o Gym (BROCKMAN et al., 2016), e que servem de base para futuros trabalhos com temas semelhantes.

Abordagens e publições deste tipo são comumente conhecidas pela comunidade de IA e utilizadas como base para novos trabalhos em jogos, incluso este, que procuram desmontrar aplicações práticas e benefícios deste tipo de treinamento. Os resultados mais debatidos tem sido com base na alta capacidade de sobreposição de desempenho que os agentes estam mostrando em relação a jogadores especialistas, a exemplos, o AlphaGo (HASSABIS, 2017) ou a própria aplicação de AR em jogos do Atari, por se comportarem bem em ambientes que tem como característica a dependência de estados, característica esta encontrada em jogos, e a capacidade de generalização de implementação em que uma modelagem de sistema de AR pode ser inserida em diversos contextos e ambientes.

Tais características podem ser testadas em ferramentas de acesso aberto que disponibilizam abstrações de ambientes pré configurados que esperam apenas o acoplamento de algoritmos de treinamentos, este processo de implementação é a metodologia principal deste trabalho que visa o desenvolvimento de um modelo AR em um ambiente controlado e a avalição de seus resultados.

Objetivos

0.0.1 Objetivo Geral

Implantação de um modelo de AR, o *Q-Learning*, em uma abstração de jogo disponibilizada pela plataforma Gym, o Taxi, afim de coletar dados de treinamento do agente e compará-lo à uma implementação de treinamento randômico baseado em algumas métricas.

0.0.2 Objetivos Específicos

- Desmonstração de eficácia de desempenho das abordagens de AR em relação a abordagens com ações aleatórias.
- Mostrar que o agente é capaz de receber a recompensa máxima definida a partir de um processo de aprendizagem por esforço.
- Evidenciar o processo de aplicação, implementação e análise dos treinamentos.

Estrutura do Trabalho

O trabalho está estruturado da seguinte maneira. O capítulo 1 descreve os temas bases, o algoritmo principal, ferramentas, ambiente e trabalhos relacionados. O capítulo 2 descreve a estruturação da metodologia de desenvolvimento e a solução proposta, descrevendo o processo de implementação e as ferramentas utilizadas. O capítulo 3 descreve os resultados obtidos. O capítulo 4 descreve as considerações finais e trabalhos futuros.

1 Revisão Bibliográfica

Este capítulo apresenta conceitos necessários para a modelagem e desenvolvimento do projeto e descrição das principais ferramentas utilizadas.

1.1 Referencial Teórico

Esta seção descreve os principais conceitos teóricos usados no desenvolvimento deste trabalho.

1.1.1 Aprendizado por Reforço

Aprendizado por reforço pode ser considerada um abordagem de comportamento de máquina que visa aprender o que fazer e como mapear situações de forma a maximizar recompensas numéricas buscando a otimização de determinado agente. Ela combina duas áreas, a programação dinâmica e o aprendizado supervisionado (HARMON; HARMON, 1997), buscando suprir necessidades operacionais em problemas específicos, a exemplos, suportar as complexidades de modelos dinâmicos como problemas com grande estrutura de Processo de Decisão de Markov(do Inglês, *Markov Decision Process - MDP*) ou resolver problemas em casos que não há respostas previamente tagueadas como acontece na aprendizagem supervisionada.

O agente interage com um ambiente dinâmico usando a abordagem de tentativa e erro (HARMON; HARMON, 1997). Ele foca principalmente nas atualizações de suas ações por meio de recompensas tentando descobrir uma boa política a partir das experiências (FIDLER, 2016), já que não há o conhecimento prévio de informações do ambiente, mas apenas informações sobre as possíveis ações e estados. Um exemplo prático seria um agente em um labirinto tentando encontrar o caminho único ou mínimo. Como o agente não possui informações sobre o ambiente e nem informações sobre possíveis melhores ações a se realizar, ele deve descobri-las explorando o labirinto e assim, o sistema se atualiza nesse processo de exploração podendo ser entendida como as tentativas e as relações de ações e estados em que recebe-se um recompensa negativa, seriam os erros, e com ele o agente aprende a procurar novos caminhos.

Um dos desafios que surgem neste processo é o *trade-off* entre *exploration* e *exploitation*. A relação destes termos está dada no fato de que deve-se haver preferência de ações que foram tentadas no passado e a partir delas foram descobertas a eficácia deste ciclo em produzir recompensa (SUTTON, 2017). A discussão está no fato de que o agente deve ter repetição na exploração para obter recompensas, mas para isso, é preciso passar por

novas ações. Richard S. Sutton e Andrew G. Barto resumem a ideia (SUTTON, 2017), o agente deve explorar o que já fora conhecido para obter recompensa, mas também realizar novas explorações para fazer melhores seleções de ações no futuro. O dilema é que nem a *exploration* e nem a *exploitation* podem ser realizadas exclusivamente sem falhar na tarefa. O agente deve tentar uma variedade de ações e favorecer progressivamente aquelas que parecem ser as melhores.

Um exemplo prático de *exploration* e *exploitation* pode ser, novamente, o caso de um agente em um labirinto. Há pontos do labirinto que existam objetos que aumentem consideravelmente a recompensa, a ação de *exploration* faz com que o agente encontre esse ponto e a ação de *exploitation* faz com que ele explore este lugar sempre, já que esta é a melhor ação, mas caso um dos pontos de alta recompensa seja um caminho que não chegue a saída, o agente poderá ficar preso e não conseguir chegar ao objetivo. Por isso, ele deve ser implementado de forma que realize ações aleatórias algumas vezes para medir novos caminhos.

Há 3 partes fundamentais que fazem parte de um problema de AR (HARMON; HARMON, 1997):

- Ambiente: De forma geral, o ambiente aqui discutido é referente a um campo artificial que fornece recursos ao treinamento do sistema AR em que nele interage para aprendizagem. Esta interação possui algumas características formais, o sistema de AR deve ao menos ter acesso parcial observável sobre o ambiente, em que as observações podem vir na forma de leituras de sensores, como as entradas de um controle, descrições simbólicas, representações de dados reais, ou possivelmente situações “mentais”, a exemplo, certa situação em que um agente está perdido e deve mudar o foco da atenção. Em um cenário perfeito, sendo este aqui colocado como a leitura perfeita de todas as informações passadas pelo ambiente, o sistema AR escolherá ações baseadas em estados verdadeiros. Este caso ideal é uma condição necessária para grande parte da teoria associada (HARMON; HARMON, 1997).

- Função de Reforço: Para cada ação realizada pelo agente há um estado associado em que esse par de conceitos se relaciona a uma dada recompensa. Esse processo leva o sistema AR a aprender ações que irão maximizar os resultados das recompensas a longo prazo baseado em um objetivo, sendo este definido usando o conceito de uma função de reforço. Existem algumas abordagens comumente usadas para a construção desta função que são próprias para determinados contextos (HARMON; HARMON, 1997), neste trabalho será discutida as abordagem de caminho mínimo associada ao jogo Taxi.

- A Função de Valor: A função de valor é o que faz uma estimativa da qualidade e da utilidade do estado atual de um agente e é definida com relação a formas particulares

de agir por meio das políticas (do Inglês, *policy*), sendo realizado um mapeamento probabilístico entre estados e possíveis ações (SUTTON, 2017).

1.1.2 Q-Learning

Segundo Watkins (1989), "*Q-learning* é uma forma de aprendizado de reforço sem modelo. Também pode ser visto como um método de programação dinâmica assíncrona. Ele fornece aos agentes a capacidade de aprender a agir de maneira ideal nos domínios markovianos, experimentando as conseqüências das ações, sem exigir que eles criem mapas dos domínios".

A definição dada pode ser explanada na definição deste ser um algoritmo *Off-Policy* para aprendizagem de diferenças temporais, ou seja, ele possui a capacidade de estimar as funções de valor em cada estado tendo o valor associado atualizado ao final de cada etapa do treinamento sem a necessidade de esperar a recompensa final para atualizar os pares de ação-estado e nem tendo tido a necessidade do mapa do domínio para regressão e atualização. Diferentemente do método *On-Policy*, em que as atualizações de valores são realizadas sempre após uma ação determinada por um política, o método *Off-Policy* realiza estimativas utilizando ações hipotéticas, ou seja, podem ser aprendidas diferentes políticas de comportamento e estimativa.

Esses tipos de política permitem o balanceamento do *trade-off* entre *exploration* e *exploitation*, já elucidados, em que o *Off-Policy* pode separar a *exploration* do controle, dando a oportunidade do agente aprender técnicas sem necessariamente ter visto elas no treinamento e tirando a necessidade de sempre realizar *exploitation* no que já fora aprendido, sendo assim, com um certo nível de treinamento sob qualquer política, o algoritmo converge com probabilidade 1 de aproximação entre a função de valor e a política de destino, ou seja, é aprendido a política ideal mesmo em um cenário exploratório ou aleatório (EDEN ANTHONY KNITTEL, 2001b).

A equação usada no *Q-Learning* tem como base a equação de *Bellman* e o processo de decisão de *Markov*. A teoria para a equação de *Bellman* foi criada para aludir os problemas matemáticos decorrentes do estudo de processos decisórios em estágios variados (BELLMAN, 1954), ou seja, apoiar o agente em suas escolhas de ações em determinados estados. Em vias de explicação da equação, s representa estado, a representa ação e R reprenta recompensa. O intuito é guardar o valor do estado atual afim de criar um caminho com valores máximos. O valor atual é atualizado com base no retorno máximo da soma entre a recompensa atual, tendo várias possibilidades de ações, e o valor atrelado ao próximo estado. O próximo estado deve ser multiplicado por um valor de desconto para que o agente não armazene o mesmo valor para todos os estados (BELLMAN, 1954). Se

encontra na fórmula abaixo:

$$V(s) = \max(R(s,a) + \gamma \times V(s')) \tag{1.1}$$

O processo de decisão de *Markov*, conhecido também como cadeia de *Markov* controlada, constitui a estrutura básica para sistemas de controle dinâmico que evoluem de maneira estocástica (ALTMAN, 1999). Ele fornece uma estrutura matemática em ambientes que os resultados variam entre aleatórios e controlados. No treinamento de um agente por AR, principalmente em jogos digitais, em certos momentos as ações do agente serão aleatórias por conta de outros agentes ou até mesmo por parâmetros de aleatoriedade adicionados. Sendo assim, cada possível futura ação tem sua própria probabilidade de ocorrer, a exemplo, um certo agente tem a possibilidade de 3 ações em determinado estado, então cada novo estado tem certa probabilidade de acontecer. Essa adaptação de contexto para a equação de *Bellman* é resolvida, em explicação simplista, adicionando um somatório dos possíveis próximos valores vezes a probabilidade de cada um (BELLMAN, 1957). Se encontra na fórmula abaixo:

$$V(s) = \max(R(s,a) + \gamma \times \sum P(s,a,s') \times V(s')) \tag{1.2}$$

O valores armazenados na tabela do *Q-Learning*, conhecida como *Q-Table*, são chamados *Q-Value*, em que neles são guardados os valores referentes a fórmula anterior (WATKINS; DAYAN, 1992), sendo assim, tudo que está dentro de max pode ser entendido como os valor do próximo estado, já que o valor atual é o máximo entre os próximos estados vezes o fator de desconto:

$$Q(s,a) = \gamma \times \max(Q(s',a')), \tag{1.3}$$

Mas como já fora comentado, o *Q-Learning* realiza as atualizações de valores por meio do *Off-Policy* e de diferenças temporais, em que a diferença temporal é entendida como o valor atual após atualização menos o valor antes da atualização (SUTTON, 1988; EDEN ANTHONY KNITTEL, 2001b; SUTTON, 2017). Sendo assim, cada novo *Q-Value* a ser guardado é igual ao *Q-Value* já armazenado mais a recompensa vinculada somada ao valor que se deseja a partir dos próximos estados, ou seja, somada a fórmula anterior aplicando-a a diferença temporal. É adicionado uma taxa de aprendizagem para que diferencie os pesos temporais dos valores da equação (EDEN ANTHONY KNITTEL, 2001a; SUTTON, 2017), então:

$$Q(s,a) = Q(s,a) + \alpha \times (r + \gamma \times \max_{a}(Q(s',a') - Q(s,a))), \tag{1.4}$$

Segue um pseudo-código da implementação do *Q-Learning*.

```
1 Inicializa a matriz de ações e estados
2 Observa o estado inicial
3 while não chegar ao objetivo do
4 |   Faça um ação
5 |   Pegue a recompensa e o novo estado
6 |   Aplique a Equação 1.4
7 |   Atualiza o estado
8 end
```

1.1.3 Gym

O *Gym* (BROCKMAN et al., 2016) é um conjunto de ferramentas desenvolvido pela *OpenAI* que provê ambientes capazes de suportar a implementação de qualquer algoritmo de aprendizagem por reforço dando auxilio no desenvolvimento e comparação entre os próprios algoritmos. Ele não espera uma estrutura pré combinada de agente e é compatível com qualquer biblioteca de computação numérica (OPENAI, 2018). Sua biblioteca é uma coleção de ambientes que são utilizados para testes dos algoritmos por meio da aplicação de tarefas, sendo estas a interação agente-ambiente. As funções fornecidas caracterizam uma interface compartilhada permitindo a construção de algoritmos gerais. Nele há funções prontas que fornecem recursos necessários para o treinamento de um agente, como os *timesteps*, recompensas, ações e estados possíveis, observações do ambiente e até mesmo informações para *debugging*.

A principal justificativa para o uso desta ferramenta é descrita na própria documentação (OPENAI, 2018), a falta de padronização de ambientes em diversos estudos em que pequenas mudanças na definição do problema, como a função de recompensa ou o conjunto de ações, podem alterar drasticamente a dificuldade de uma tarefa.

1.1.4 Taxi - Ambiente de Texto

A ideia do jogo é fazer com que um taxi pegue um passageiro em um ponto A e o leve a um ponto B por meio do menor caminho. O ambiente visual fornece 4 localizações, representações visuais de obstáculos e representação do taxi. O retângulo representa o carro, onde caso esteja em cor amarela significa que está vazio e caso esteja na cor verde, há um passageiro dentro. Os dois pontos representam as divisões das ruas e os traços representam paredes em que o carro não deve atravessar.

As 4 localizações, representadas por letras, mostram os possíveis pontos de partida e destino do passageiro, sendo a letra em cor azul mostrando a localização do passageiro e

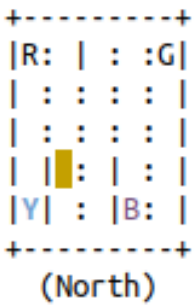

Figura 1 – Representação visual em texto do jogo

em cor roxa, o destino. O carro pode se mover na vertical e na horizontal e realizar ações de pegar ou soltar o passageiro.

1.2 Trabalhos Relacionados

Aplicando *Q-Learning* em jogos digitais (BRENDEL, 2018) é um trabalho que descreve e analisa os resultados da aplicação do algoritmo no jogo *Table Heroes*. O jogo se resume no confronto entre dois times rivais e termina quando um dos dois estiver com todos os jogadores derrotados. O algoritmo fora implementado em sua forma padrão, assim como neste trabalho. O ambiente passa para o agente sua posição atual em coordenadas x e y e se há inimigos adjacentes sem saber a posição exata. O agente pode se mover para o norte, sul, leste e oeste, e atacar nas mesmas direções. O número total de estados é duas vezes o número de blocos defendidos pelos inimigos.

São utilizadas duas métricas para avaliar a aplicação do algoritmo, avaliação de ponto de volta, que define que o ponto de volta é quando o número de vitórias excederá em 10 o número de derrota, e avaliação de lapso de tempo, que procura analisar apenas uma execução do programa ao longo de um grande número de gerações, permitindo observar quanto tempo o agente leva para atingir seu estado máximo de aprendizado.

Jogando Atari com Aprendizagem de Esforço Profundo (MNIH et al., 2013) é considerado o primeiro trabalho a aprensentar um modelo de aprendizagem por esforço pronfundo podendo ser visto como um trabalho chave para a evolução da aplicação do *Q-Learning* em ambientes mais complexos. Publicado pela *DeepMind*, o objetivo é aprender políticas de controle a partir de AR usando uma variante do *Q-Learning* para treinar uma rede neural convolucional.

A aplicação do algoritmo foi realizada em jogos do *Atari* 2600 implementados no ambiente de aprendizagem Arcade (ALE), em que as entradas visuais tem alta dimensionalidade (vídeo RGB de 210 160 RGB a 60Hz). O objetivo foi criar um único agente capaz de aprender a jogar o maior número possível de jogos sem receber informações prévias do ambiente aprendendo pela entrada visual, recompensas, ações e estados disponíveis.

Nos dois trabalhos houve resultados esperados de aprendizagem generalizada e adaptação a ambientes desconhecidos, revelando a evolução esperado dos agentes por meio da aplicação do *Q-Learning* e suas variações.

2 Metodologia

Neste capítulo serão apresentados os valores dos paramêtros para o algoritmo e ambiente, comportamento do ambiente, configuração de ambiente de teste, arquitetura do agente e métricas para avaliação do algoritmo. Os algoritmos implementados se encontram em anexo.

2.1 Parâmetros

A leitura do jogo é feita de forma textual em que o ambiente passa as informações ao agente por interfaces númericas, sendo este processo melhor detalhado no próximo tópico. São possíveis 4 ações, norte, sul, leste e oeste, todas resumidas em movimentação do carro, e sendo um grid 5x5 com 4 possíveis localizações e 4 possíveis posições do passageiro mais 1 que seria dentro do taxi, são possíveis 5x5x5x4 ou 500 estados em que o agente pode se encontrar.

As recompensas são definidas pela própria ferramenta *Gym*, em que são definidos 20 pontos para quando o agente chegar ao objetivo que é a entrega do passageiro, -10 pontos, caso o taxi realize a ação de soltar nos estados em que não possui o passageiro ou em que não está na posição correta tendo-o ou não, -10 pontos, caso realize a ação de pegar nos estados em que não está na posição do passageiro ou já o tem, e -1 ponto para cada ação de movimento, a fim de forçar o agente a realizar a tarefa na menor quantidade de passos possíveis.

Tomando como base a Equação 1.4 para implementação do algoritmo, seus parâmetros foram definidos da seguinte maneira:

- O α é a taxa de aprendizado que deve ser configurada entre 0 e 1, em que configurar para 0 significa que os valores-Q nunca são atualizados, e, portanto, nada é aprendido e definir um valor alto como 0,9 significa que o aprendizado pode ocorrer rapidamente (EDEN ANTHONY KNITTEL, 2001a).

- O γ, fator de desconto, também definido entre 0 e 1 em que configura o fato de que recompensas futuras valem menos do que recompensas imediatas e faz com que elas sejam diferentes para cada estado. Próximo de 1 configura que as recompensas futuras tem alto peso, o que é necessário para o encontro de um caminho mínimo, sendo o valor definido aqui como 0,9.

O agente inicia o processo de treinamento sem conhecimento prévio do ambiente e o explora utilizando de ações de movimento e das atualizações da *Q-Table*. Inicialmente,

e durante as iterações, ele recebe informações sobre as possíveis ações e estados e ao se movimentar recebe *feedbacks* em forma de recompensas e detalhes do novo estado em que se encontra.

2.2 Ciclo de Treinamento e Comportamento do Ambiente

A ferramenta *Gym* fornece uma abstração do ambiente totalmente configurado para receber diferentes estilo de interface do agente. Nele são guardados diversos tipos de ambiente, sendo para este trabalho, o principal, o ambiente *Toy Text*, ambientes de simulação em forma de texto.

Após cada ação do agente, o ambiente *Gym* retorna 4 valores em que esse retorno pode ser entendido como a comunição em forma de texto do ambiente descrito, *observation* do tipo Object, que é uma representação específica do ambiente, como exemplo, o estado atual do jogo, *reward* do tipo *float*, quantidade de recompensa alcançada pela ação anterior, *done* do tipo *boolean*, diz se um episódio terminou para que, então o ambiente possa ser resetado, e *info* do tipo *dict*, dados úteis para depuração em que pode ser usado para a coleta de métricas já que armazena as probabilidades brutas relacionadas a ultima alteração de estado.

Serão realizados alguns ciclos de treinamentos gradativos para encontrar o ponto em que o agente tenha no mínimo 90% de vitórias, pois com a quantidade de episódios encontradas deste valor, sendo um episódio neste trabalho entendido pela inicialização do ambiente e das variáveis que guardam as penalidades e recompensas adquiridas em no máximo 200 ações ou passos realizadas pelo agente, valor este definido pela própria ferramenta no retorno da variável *done*, os testes subsequentes serão realizados.

Um treinamento é entendido pela inicialização da *Q-Table* e dos parâmetros de treinamento e termina com o resultado final da *Q-Table* e sua política. Caso o agente ganhe o jogo antes das 200 iterações, o *loop* é quebrado. Dentro da iteração ações/passos, ocorre a mudança contínua da *Q-Table* por conta da utilização das recompensas e dos estados retornados que atualizam os *Q-Values*.

Ao final de cada treinamento é coletado a quantidade de vitorias, os valores de penalidades, quantidade de estados que se repetiram e quantidades de ações, todos por episódio.

2.3 Métricas

As métricas escolhidas foram:

- Comparação gradativa da porcentagem de vitórias sobre o total de episódios.

- Comparação do nível de dimunição de penalidades por episódios.
- Comparação do nível de dimunição de passos por episódios.

Para a análise das métricas fora implementado um agente que realiza o treinamento com ações aleatórias, já que segundo Justin Francis (FRANCIS, 2017), uma parte essencial da avaliação do desempenho de qualquer agente é compará-lo a um agente completamente aleatório. Ao final dos testes, os resultados das duas implementações foram comparadas.

2.4 Configuração do Ambiente

A implementação do algoritmo e realização do treinamento foram feitas em uma máquina virtual com as seguintes especificações:

Tabela 1 – Especificações da máquina virtual

Hardware	**Especificação**
Memória RAM	4096 MB
Processador	3 núcleos - Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz
Disco Rígido	50 Gb

Para a preparação do ambiente de máquina foram instalados, utilizando do gerenciador de pacotes Pip, as bibliotecas Numpy, na versão 1.15.4, para o processamento da funções na matriz *Q-Table*, e a Gym, na versão 0.10.9, para uso das funções de ambiente de treinamento. A linguagem de desenvolvimento é o Python 3.5+, pois é o necessário para integração com o *Gym* (OPENAI, 2018).

Para o uso das bibliotecas, após a instalação, é necessária a realização de suas importações no arquivo Python a ser desenvolvido o algoritmo. No intuito de integração com a plataforma, usa-se a função *gym.make* passando como parâmetro o nome do ambiente que deve ser carregado. Essa função retorna todas as informações e ações possíveis sobre o ambiente. A partir de uma variável, em que aqui será identificada como *env*, que guarda tal retorno, é possível usar da quantidade de estados e ações possíveis para inicializar e atualizar dinamicamente a *Q-Table*, realizar a movimentação do agente de um estado a outro e obter todas as informações de treinamento.

Sendo assim, a configuração de ambiente de máquina se resume a instalação das bibliotecas e as informações de ambiente são todas usadas pela importação e carregamento via *Gym*.

A interface implementada resume-se a importação do ambiente via plataforma *Gym*. Os paramêtros de treinamento como o α e o γ são definidos de forma global e então, deve ser realizado um *loop* do tamanho da quantidade de episódios escolhidos ou

até que o agente alcance o objetivo e que dentro tem-se o uso do da equação do *Q-Learning* para atualizar dinamicamente a *Q-Table*.

3 Resultados

Nesta seção são apresentados os resultaodos em forma de comparações entre as duas abordagens, *Q-Learning* e randômica. As subseções estão divididas pelas 3 métricas levantadas.

Para a realização dos testes em prol das análises das métricas, foram efetuados alguns treinamentos acrescentando gradativamente a quantidade de episódios por treinamento. O processo teve começo em 60 episódios até que o sistema passasse da média de 90% de vitórias por episódio, mostrando então, que o agente está moldado de uma política forte de ações, sendo colocada neste trabalho como próximo a 100% de vitórias. Assim, obtem-se um valor de episódios considerável para os treinamentos.

Tabela 2 – Média de Vitórias por Episódio

Qtd episódios	**Média de Vitórias**
60	46.7%
80	57%
100	64.1%
150	76.87%
300	88.67%
350	90%
400	91%

Finalizado o processo acima, o último valor da tabela, 400 episódios, foi escolhido como a quantidade para os testes seguintes.

Os treinamentos, com o uso da abordagem de AR e randômica demoraram, respectivamente, 1,330 e 2,434 segundos, em média. A complexidade dos algoritmos de AR e randômico desenvolvidos é de O(e*d), sendo e a quantidade de episódios e d a quantidade de passos por episódio.

3.1 Porcentagem de Vitórias

Como já pôde ser percebido na análise da tabela anterior, o agente treinado pelo o algoritmo *Q-Learning*, como esperado, gradativamente, aumenta sua porcentagem de vitórias ao passar dos episódios de treinamento. Este aumento acontece por conta das atualizações dos *Q-Values* que passam a fornecer a política ideal de vitória ao agente, já no caso do agente randômico, como não há nenhuma política a ser alcançada, os resultados são sempre baixo e aleatórios.

Tabela 3 – Comparação - Média de Vitórias por Episódio

Qtd episódios	**Média de Vitória (Q-Learning)**	**Média de Vitória (Aleatório)**
150	77%	5.7%
300	89%	6%
350	90%	5%
400	91%	4.8%

3.2 Dimunição de penalidades

A forma como o agente é forçado a achar o caminho mais curto é pelas penalidades inseridas quanto em ações ditas erradas tanto em movimentação. Sendo assim, analisando o gráfico abaixo, é visto que no início do treinamento, por conta da exploração de novos estados, o agente com abordagem *Q-Learning* demonstra um comportamento, em relação a ganho de penalidades, semelhante a um modelo randômico. Deve ser colocado que, mesmo sendo um modelo com resultados aleatórios, após a realização de diversos treinamentos, o modelo randômico sempre manteve em proporções altas e semelhantes ao próximo gráfico.

Percebe-se que após 100 iterações, o algoritmo *Q-Learning* já possui queda considerável sobre as penalidaes, em média de 60%, o que coincide em média com a faixa de 64% de vitórias nos episódios. Os responsáveis por esse rápido treinamento são os parâmetros, taxa de aprendizado e fator de desconto, que foram definidos com altos valores. No final, as penalidades se mantêm em fluxo quase que contínuo e baixo.

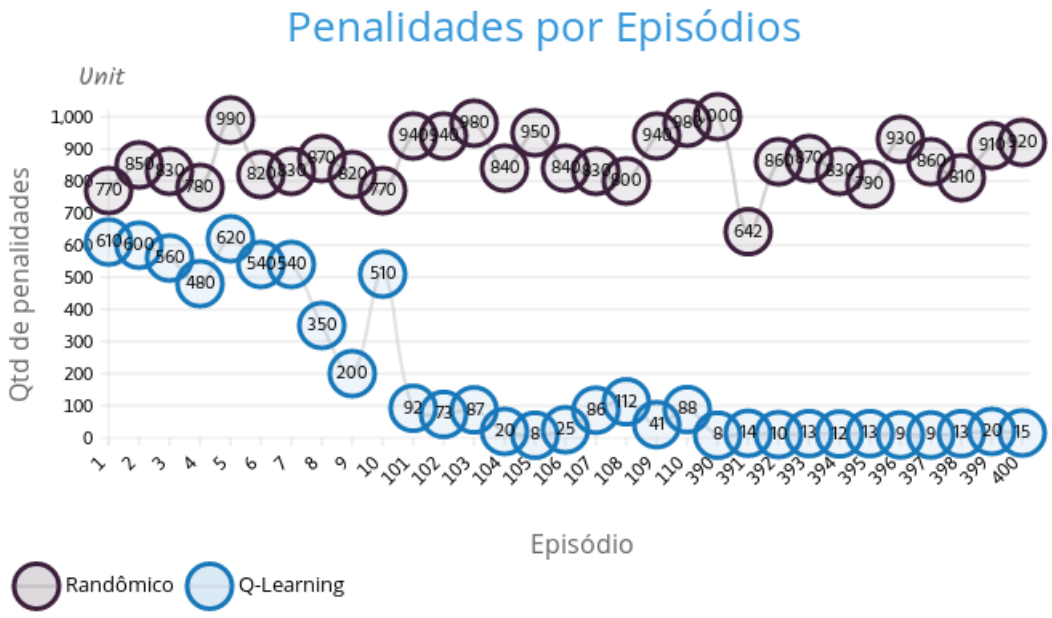

Figura 2 – Penalidades por Episódio

3.3 Dimunição de passos por episódios

O sistema AR realizou explorações repetitivas em proporções consideráveis até a faixa de 100 à 150 episódios, como pode ser analisado no gráfico seguinte de repetições, demonstrando que quanto mais a política é modelada mais o agente se movimenta de forma direta e sem a necessidade de exploração de tantos estados, mesmo assim, é uma fase do treinamento que ainda oscila com novas explorações. No caso do agente randômico, como já pôde ser visto que sua porcentagem de vitórias é baixo, então o gráfico de passos tende a ser sempre uma linha reta no máximo.

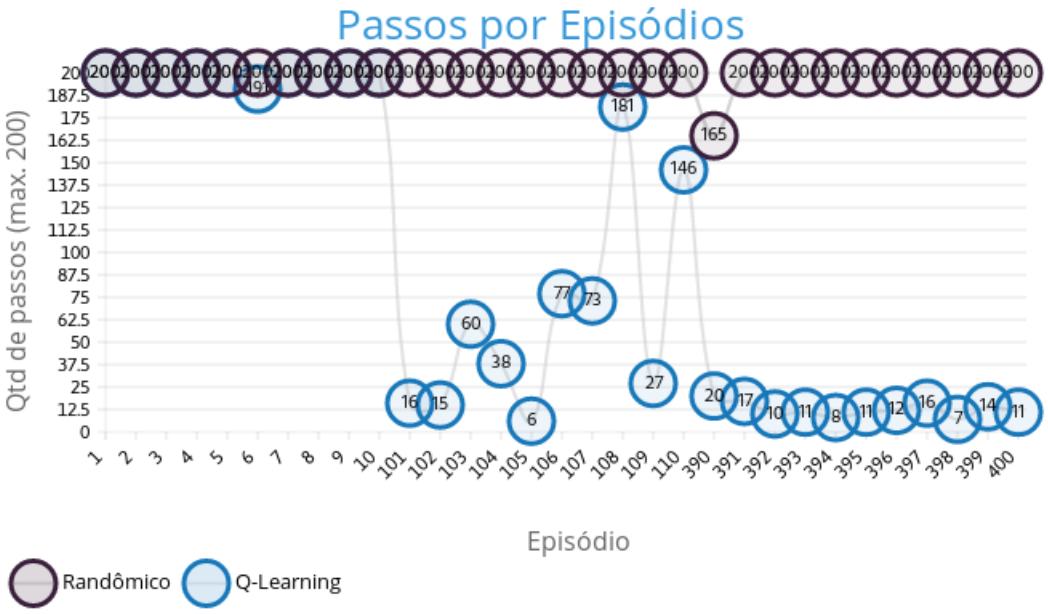

Figura 3 – Passos por Episódio

Como pode ser visto, ao decorrer do treinamento com uso da abordagem de AR o parâmetro d da complexidade passaria a convergir para o tamanho médio do intervalo da distância mínima e da distância máxima em que o taxi inicia e seu ponto final. E a complexidade do modelo randômico se mantém.

4 Conclusão

Este trabalho explanou as características, processo de implementação e análise comparativa da abordagem *Q-Learning* sobre o ambiente de jogo com interface em texto da plataforma Gym. O sistema proposto teve toda sua modelagem baseada apenas na implementação dos treinamentos e integração com a plataforma. Após as definições dos parâmetros para a equação base da abordagem, os treinamentos foram realizados buscando retornar todos os dados sobre recompensas, penalidades, episódios, quantidades de ações necessárias e, principalmente, se o agente conseguiu chegar a uma política de treinamento que o fizesse ganhar o jogo facilmente dutante iterações subsequentes.

Os resultados mostraram-se satisfatórios com o que os parâmetros α e γ propunham e com a forma de atribuição de valores e pesos adquiridas pelo relacionamento de estados, ações e recompensas dentro do *Q-Table*.

Os treinamentos realizados mostraram rápida performance por conta dos altos valores atribuidos a taxa de aprendizagem e ao fator de desconto. O agente se manteve em aprendizagem constante revelada pela queda contínua das penalidades até o limite, pela quantidade de vitórias em certo intervalo de episódios e pela diminuição gradativa da média de quantidade máxima de ações necessárias para ganhar o jogo.

No intuito de incentivo sobre estudos focados na área de aplicação de aprendizado de máquina em jogos, principalmente pelas diversas abordagens de aprendizagem por esforço, propomos os seguintes trabalhos futuros de forma que possam incorporar novas análises de desempenho no sistema proposto:

- Implementação do *Q-Learning* em outros jogos no ambiente de texto, a fim de testar a generalização da abordagem;
- Implementação do *Deep Q-Network*, variação do *Q-Learning* que utiliza redes neurais convolucionais, em cima dos ambientes Atari e sua comparação com abordagens Neuroevolutivas visando a análise de derivados do próprio *Q-Learning*;

Referências

ALTMAN, E. *Constrained Markov Decision Processes*. [S.l.]: Chapman and Hall, 1999. Citado na página 28.

BELLMAN, R. The theory of dynamic programming. *Bull. Amer. Math. Soc.*, American Mathematical Society, v. 60, n. 6, p. 503–515, 11 1954. Disponível em: <https://projecteuclid.org:443/euclid.bams/1183519147>. Citado na página 27.

BELLMAN, R. A markovian decision process. *Journal of Mathematics and Mechanics*, v. 6, n. 5, p. 679–684, 1957. Disponível em: <http://www.jstor.org/stable/24900506>. Citado na página 28.

BRENDEL, J. R. B. Guilherme de Q. Application of q-learning in a digital game. *SBGames*, 2018. Citado na página 30.

BROCKMAN, G. et al. *OpenAI Gym*. 2016. Citado 2 vezes nas páginas 23 e 29.

EDEN ANTHONY KNITTEL, R. v. U. T. *Reinforcement Learning - Algorithms*. 2001. Disponível em: <https://www.cse.unsw.edu.au/~cs9417ml/RL1/algorithms.html>. Citado 2 vezes nas páginas 28 e 33.

EDEN ANTHONY KNITTEL, R. v. U. T. *Reinforcement Learning - TD Learning*. 2001. Disponível em: <https://www.cse.unsw.edu.au/~cs9417ml/RL1/tdlearning.html#aselection>. Citado 2 vezes nas páginas 27 e 28.

FIDLER, S. Csc 411: Lecture 19: Reinforcement learning. Class based on Raquel Urtasun & Rich Zemel's lectures. 2016. Citado na página 25.

FRANCIS, J. *Introduction to reinforcement learning and OpenAI Gym*. 2017. Disponível em: <https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym>. Citado na página 35.

HARMON, M. E.; HARMON, S. S. Reinforcement learning: A tutorial. Jan 1997. Citado 2 vezes nas páginas 25 e 26.

HASSABIS, D. S. D. *AlphaGo Zero: Learning from scratch*. 2017. Disponível em: <https://deepmind.com/blog/alphago-zero-learning-scratch/m>. Acessado em: 20 de Novembro de 2018. Citado na página 23.

KONAR, A. *Computational intellingence: principles, techniques, and applications*. [S.l.]: Springer, 2005. Citado na página 23.

MNIH, V. et al. Playing atari with deep reinforcement learning. Jan 2013. Citado 2 vezes nas páginas 23 e 30.

OPENAI. *Getting Started with Gym*. 2018. Disponível em: <https://gym.openai.com/docs/>. Citado 2 vezes nas páginas 29 e 35.

SUTTON, A. G. B. R. S. *Reinforcement Learning: An Introduction*. second. [S.l.]: The MIT Press, 2017. Citado 4 vezes nas páginas 25, 26, 27 e 28.

SUTTON, R. S. Learning to predict by the methods of temporal differences. *Mach. Learn.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 3, n. 1, p. 9–44, ago. 1988. ISSN 0885-6125. Disponível em: <http://dx.doi.org/10.1023/A:1022633531479>. Citado na página 28.

WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — King's College, Cambridge, UK, May 1989. Disponível em: <http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf>. Citado na página 27.

WATKINS, C. J. C. H.; DAYAN, P. Q-learning. In: *Machine Learning*. [S.l.: s.n.], 1992. p. 279–292. Citado na página 28.

Anexos

ANEXO A – Código - *Q-Learning*

```
import gym
import numpy as np
import random

env = gym.make('Taxi-v2') # load the environment

# print('State space', env.observation_space.n) # Amount of available s
# print('Action space', env.action_space.n) # Amount of available actio

number_episodes = 400
total_train = 1

file = open('../results/text_results.txt', 'w')

for aux in range(total_train):
  # Initialize the Q-Table with state x action
  q_table = np.zeros([env.observation_space.n, env.action_space.n])

  total_loss = []
  total_wins = 0
  count_steps = []
  total_state_remained = []
  alpha = 0.9
  gamma = 0.9
  for i in range(number_episodes):
    # Reset the environment to initialize a new ep
    initial_state_of_ep = env.reset()

    # print('\n========== EPISODE START ==========\n')
    # print(f'The initial episode state are: {initial_state_of_ep}')

    is_done = False
    loss, state_remained = 0, 0
    current_state = initial_state_of_ep
    count = 0
```

```
while not is_done:
  # env.render() #-> use this command to see the game runing

  # To see the current state
  # print(f'\nState BEFORE the action: {current_state}')

  # Select the highest Q Value for the current state
  action = np.argmax(q_table[current_state])

  # print(f'Action {action}')

  new_state, reward, is_done, info_for_debug = env.step(action)
  # env.render() #-> use this command to see the game runing

  # values = [new_state, reward, is_done, info_for_debug]
  # messages = ['New state:', 'Reward of the current state:',
  #             'Has the agent finished the task ?', 'Info for debuggin
    # print(f'State AFTER the action: {new_state}')

  # Update Q Table
  q_table[current_state, action] += alpha * (reward + gamma * np.max(q_

  # The state remained
  if current_state == new_state:
    state_remained += 1

  # Loss 10 points if the car is empty and perform
  # an action involving the passenger
  if reward == -10:
    loss += 10

  if reward == 20:
    # input('========================================= WIN! Press a key
    #env.render() #-> use this command to see the game runing
    total_wins += 1

  # To see informations about each action taken
  # for v, m in zip(values, messages):
```

```
        #    print('{} {}'.format(m, v));
        # print('\n')

        loss += 1
        current_state = new_state
        count += 1

    total_loss.append(loss)
    count_steps.append(count)
    total_state_remained.append(state_remained)

    # print(f'Total loss of this episode {loss}')
    # print(f'\n========== EPISODIO {i + 1} TERMINOU ==========\n')

  file.write(f'========== TREINO {aux + 1} TERMINOU ==========\n')
  file.write(f'O agente ganhou o jogo {total_wins} vezes\n\n')
  file.write(f'Lista de penalidades {total_loss}\n\n')
  file.write(f'Lista de passos {count_steps}\n\n')
  file.write(f'Lista de estados repetidos por episodio [1-400] - max(20

file.close()
```

ANEXO B – Código - Ações aleatórias

```
import gym
import numpy as np

env = gym.make('Taxi-v2') # load the environment

# print('State space', env.observation_space.n) # Amount of available s
# print('Action space', env.action_space.n) # Amount of available actio

number_episodes = 400
total_train = 1

file = open('../results/random_text_results.txt', 'w')

for aux in range(total_train):
  total_loss = []
  total_wins = 0
  count_steps = []
  total_state_remained = []
  for i in range(number_episodes):
    # Reset the environment to initialize a new ep
    initial_state_of_ep = env.reset()

    # print('\n========== EPISODE START ==========\n')
    # print(f'The initial episode state are: {initial_state_of_ep}')

    is_done = False
    loss, state_remained = 0, 0
    current_state = initial_state_of_ep
    count = 0

    while not is_done:
      # env.render() #-> use this command to see the game runing

      # To see the current state
      # print(f'\nState BEFORE the action: {current_state}')
```

```
    action = env.action_space.sample()

    # print(f'Action {action}')

    new_state, reward, is_done, info_for_debug = env.step(action)
    # env.render() #-> use this command to see the game runing

    # values = [new_state, reward, is_done, info_for_debug]
    # messages = ['New state:', 'Reward of the current state:',
    #             'Has the agent finished the task ?', 'Info for debuggin
      # print(f'State AFTER the action: {new_state}')


    # The state remained
    if current_state == new_state:
      state_remained += 1

    # Loss 10 points if the car is empty and perform
    # an action involving the passenger
    if reward == -10:
      loss += 10

    if reward == 20:
      # input('========================================= WIN! Press a key
      #env.render() #-> use this command to see the game runing
      total_wins += 1

    # To see informations about each action taken
    # for v, m in zip(values, messages):
    #   print('{} {}'.format(m, v));
    # print('\n')

    loss += 1
    current_state = new_state
    count += 1

  total_loss.append(loss)
  count_steps.append(count)
```

```
        total_state_remained.append(state_remained)

        # print(f'Total loss of this episode {loss}')
        #print(f'\n========== EPISODIO {i + 1} TERMINOU ==========\n')

    file.write(f'========== TREINO {aux + 1} TERMINOU ==========\n')
    file.write(f'O agente ganhou o jogo {total_wins} vezes\n\n')
    file.write(f'Lista de penalidades {total_loss}\n\n')
    file.write(f'Lista de passos {count_steps}\n\n')
    file.write(f'Lista de estados repetidos por episodio [1-400] - max(20

file.close()
```